C# Lambda İfadeleri (Lambda Expressions)

yazilim.cevapsitesi.com/Makaleler/2/cSharp-lambda-ifadeleri-lambda-expressions

Yazılım Makaleleri

Lambda İfadeleri, sadeleştirilmiş anonim (isimsiz) fonksiyonlardır. Bu fonksiyonlar matematikteki ve bilgisayar bilimlerindeki Lambda Calculus'un C# ve Visual Basic uygulamasıdır.

Lambda İfadeleri kullanarak parametre geçilebilen ve değer döndüren isimsiz yerel fonksiyonlar oluşturabilirsiniz. Bu ifadeler genelde basit işlemleri bildirmekte kullanılabilir ve LINQ sorgularının yazımında çok işe yararlar. LINQ konusunu bu yazıdan sonra incelemenizi tavsiye ederiz.

Örnek :

```
1. n => n * n
```

Bir Lambda İfadesi oluşturmak örnekte olduğu gibi eğer varsa sol tarafta parametre bildirimini, lambda operatöründen (=>) sonra ifadeyi veya kod bloğunu sağ tarafa yazın.

Üstteki ifadenin anlamı şudur;

Bir parametre al ve bu parametreyi n kabul et (soldaki n), n'yi kendisiyle çarpıp sonucu döndür (sağdaki n * n). Bu fonksiyon eğer anonim (isimsiz) fonksiyonlara yabancı iseniz ismi (kare) olması ve tip bildirimleri (int) dışında aşağıdakiyle aynıdır.

```
1. int kare(int n)
2. {
3.     return n * n;
4. }
```

Aynı işlemi yapan fonksiyon anonim olarak da şu şekilde tanımlanır;

```
1. delegate (int n)
2. {
3.     return n * n;
4. }
```

Eğer bu tanımlamadan delegate ifadesini çıkarırsanız bir Lambda İfadesi elde eder ve kullanabilirsiniz. Artık bir Lambda İfadesi olan isimsiz fonksiyonun yeni hali...

```
1. (int n)
2. {
3.     return n * n;
4. }
```

şeklinde olacaktır.

Süslü parantezleri lambda işlecine dönüştürürsek;

```
1. (int n) => return n * n;
```

biçiminde hala geçerli olan bir Lambda İfadesi elde ederiz. n * n tek bir ifade olduğundan noktalı virgülü ve return ifadesini kaldırırsak;

```
1. (int n) => n * n
```

biçimi kalır ve geçerli bir Lambda İfadesidir.

n parametresinin tipi biz vermesek bile derleyici tarafından uygun biçimde tanımlanacağından parametrenin türünü belirleyen int ifadesini de kaldırabiliriz. Bu durumda fonksiyon;

```
1. (n) => n * n
```

olur. Artık tip kullanmadığımıza ve tek parametremiz olduğuna göre baştaki parantezler de gereksizdir. Ve geriye kalır;

```
1. n => n * n
```

Görüldüğü gibi isimsiz fonksiyonlarla Lambda İfadeleri aynıdır. Şimdiye kadar gördümüz kadarıyla aralarındaki tek fark lambda fonksiyonlarının tek satır halinde olmasıdır. Tek satır halindeki Lambda İfadelerine Expression Lambdas (İfade Lambdaları) denir.

İfadenin sağ tarafında süslü parantezler ve sonunda da noktalı virgül kullanılarak çok satırlı bir Lambda İfadesi de yazılabilir. Böyle yazılan Lambda İfadelerine Statement Lambdas (Komut Lambdaları) denir.

```
1. n =>
2. {
3.     string s = "Merhaba " + n;
4.     Console.WriteLine(s);
5. };
```

Lambda İfadelerinde parametre zorunlu değildir. Aşağıdaki de geçerli bir ifadedir.

```
1. () => birFonksiyon()
```

Parametreler bir parametreden fazla olabilir. Bu durumda parametreleri parantez içinde bildiririz.

```
1. (a, b) => a > b
```

(2 parametre al, birincisi a, ikincisi b olsun. a'nın değerinin b'den büyük olup olmadığını döndür.)

Yukarıda verilen Lambda İfadelerini program koduna eklerseniz "Only assignment, call, increment, decrement, await, and new object expressions can be used as a statement." (Sadece atama, yordam çağrısı, arttırma (++), eksiltme (--), bekleme (await) ve yeni nesne ifadeleri (new) komut olarak

kullanılabilir.) şeklinde bir uyarı görürsünüz. Yani bu ifadeleri tek başına yazmanın bir anlamı yoktur ve çalışacak bir şey değildir.

Şimdi bunların nasıl kulanıldıklarından bahsedelim.

Başlangıçta da belirtildiği gibi Lambda İfadelerinin en fazla işe yaradığı yer LINQ sorgularıdır. LINQ bir kaç cümleyle anlatılamayacağından ve bilmeyenler için şu anda kafa karıştırıcı olacağından dolayı ayrıca incelenmelidir. Burada sadece LINQ'in iki kullanım şeklinden biri olan yöntem sözdizimi (method syntax) kullanılarak yazılan alttaki örnekten yararlanabiliriz (Eğer isterseniz LINQ konusundan sonra dönüp bakmak üzere örneği atlayıp sonraki paragraftan delege konusuna devam edebilirsiniz).

```
 1. namespace Console_Lambdas1
 2. {
 3.     class Program
 4.     {
 5.         static void Main(string[] args)
 6.         {
 7.             // Tüm rakamları kullanarak bir dizi tanımla.
 8.             int[] tumRakamlar = new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
 9.             // Tek rakamları seç ve yeni bir koleksiyona kopyala.
10.             IEnumerable<int> tekRakamlar = tumRakamlar.Where(x => x % 2 == 1);
11.             Console.WriteLine("Tek rakamlar :");
12.             // tekRakamlar dizisinin elemanlarını yaz.
13.             foreach (int rakam in tekRakamlar) Console.WriteLine(rakam);
14.             Console.ReadKey();
15.         }
16.     }
17. }
```

Örnekte görüldüğü gibi LinQ'in yöntem sözdizimi (.Where(...) bölümü) kullanılmış, Where seçme yöntemine bir Lambda İfadesi parametre olarak gönderilmiştir. Program, tüm rakamların dizisinden tek olan rakamları seçer ve ekrana yazar.

Örnekteki Lambda İfadesi x isimli bir parametre alır, x mod 2 işlemini çalıştırarak sonucun 1'e eşit olup olmadığını döndürür. Where seçici yöntemi, IEnumerable (Türkçesi sayılabilir) arayüzünü veya IEnumerable<T> genelleyici (generic) arayüzünü uygulayan tüm sınıflarda kullanılan, sınıfın (ki bunlar bizim örneğimizdeki int dizisi gibi elemanları bulunan List, ArrayList vb. koleksiyon sınıflarıdır) elemanlarını tek tek gezip kendisine gönderilen Lambda İfadesi uygulandığında "true" sonucu döndüren elemanları seçerek sonuç olarak yine IEnumerable türünden bir sınıfa ekler ve bu sınıfı döndürür.

LINQ dışında Lambda İfadeleri kendileri de isimsiz fonksiyonlar olduklarından fonksiyonları temsil etmede kullanılan delege (delegate) tanımlamalarında kullanılırlar. Aşağıdaki örneği inceleyin...

```
using System;
using System.Collections.Generic;
using System.Linq;
namespace Console_Lambdas2
{
    class Program
    {
        // Tamsayı döndüren ve parametre olarak 2 adet tamsayı alan bir delege tipi tanımla.
        delegate int TamsayiIslevi(int sayi1, int sayi2);
        static void Main(string[] args)
        {
            // tamsayiIslevi delegesi türünden bir delege değişken
            // tanımla (= işaretinin solundaki bölüm) ve delegeye
            // Lambda İfadesi ile tanımlanmış olan bir isimsiz
            // fonksiyonu ata (= işaretinin sağındaki bölüm).
            TamsayiIslevi tsiTopla = (x, y) => x + y;
            // Aynı delege tipini kullanarak 2. bir isimsiz
            // fonksiyon tanımla ve bir delege değişkene ata.
            TamsayiIslevi tsiCikar = (x, y) => x - y;
            // Delege değişkenleri kullanarak referansı oldukları
            // fonksiyonları çalıştır ve çıktıya yaz.
            Console.WriteLine(tsiTopla(5, 2));
            Console.WriteLine(tsiCikar(5, 2));
            Console.ReadKey();
        }
    }
}
```

Delegeler (Türkçesi temsilciler), fonksiyonların referansı (fonksiyona ulaşmak için başvurulacak kaynak) olarak kullanılırlar. Örnekte "delegate" ifadesi kullanılarak bir delege tipi tanımlanmıştır. Main yordamında ise bu tip delege türünden delege değişkenler tanımlanarak Lambda İfadesi yöntemi ile anonim fonksiyonlar delege değişkenlere atanmıştır. Bundan sonra bu delege değişkenler, kendilerine atanan anonim fonksiyonlara referans olmuştur. Sonraki satırlarda ise bu referanslar ve uygun parametreler (tsiTopla(5, 2) gibi) kullanılarak delegelerin (örneğin tsiTopla'nın) referansı olduğu fonksiyonlar (örneğin (x, y) => x + y) çalıştırılmıştır.

Delegelerin çok faydalanıldıkları bir konu da olaylardır. Aynı delegenin bir fonksiyondan fazlasına referans olabilmesi (aynı olay için çalıştıralacak birden fazla fonksiyon olabilir) ve birbirine benzer türde bir çok olay bulunması sebebiyle olay kullanımında çok işe yararlar. Bu konu mutlaka incelenmelidir.

Tek satırlı lambda ifadeleri (Expression Lambdas), birer ifade ağacıdır (Expression Tree) ve bundan dolayı bir lamda ifadesini çözümleyebilir veya oluşturabiliriz. Hatta C# ile çok satırlı ifadeleri de Expression Sınıfı kullanarak oluşturabilir ve çözümleyebiliriz. İfadeleri oluşturmak ve çözümlemek sıkça kullanılabilecek bir işlem olmasa da ifade ağaçları ve Expression Sınıfının kullanımını öğrenerek programlama bilginizi bir adım daha ileri götürmek için İfade Ağaçları (Expression Trees) ve Expression Sınıfı yazımı okumanızı tavsiye ederim.